Abdirashid Abbas

16 febbraio 2022

Relazione del laboratorio di TLN (parte 3)

Prima parte

1.

Nel primo esercizio la consegna era quella di estrarre da un foglio google tutte le definizioni di alcuni termini proposte dagli studenti del corso . I termini sono divisi secondo 2 criteri: astratto, generico, specifico e concreto. Nel corso sono stati proposti i seguenti termini:

Courage: astratto-generico

Sharpener: concreto-specifico

Apprehension: astratto-specifico

Paper: concreto-generico

Per ogni lista di definizioni di ciascun termine è stata creato una lista di dizionari che li contenga tutti, dove in ciascun dizionario come chiave è stata settato il termine in considerazione.

Dopodiché è stato sviluppato un metodo get_list_items, che prende come parametri la lista di dizionari "completa", e una stringa che indica il termine(e quindi chiave) che ci interessa. Questo metodo restituisce la lista di tutte le definizioni per quel dato termine.

Poi si effettua il prepocessing andando a rimuovere le stop words e i caratteri che non servono.

Infine come misura di similarità si è scelti l'overlap dei termini, restituendo infine una misura di similarità che è sostanzialmente il quoziente tra la lunghezza della lista overlap e la lunghezza media dei termini rilevanti per ciascuna definizione.

I risultati confermano quanto potevamo immaginare, ovvero che aumentando la genericità e l'astrattezza di un termine, otterremo un agreement inter-annotator (similarità) minore.

```
def overlap(list,string):
    overlap_similarity=0
    if string=='courage':
        tot_ann=len(list)
        average_relevant_words=[]
        avg_length=0
        len_overlap=0
        for i in list:
            for l in i.keys():
                if l=='Relevant_words':
                    if len(average_relevant_words)<1:
                        average_relevant_words.append(i.get(l))
                    else:
                        overlap=intersection(average_relevant_words,i.get(l))
                        #print(overlap)
                        if len(overlap)>len_overlap:

                            len_overlap=len(overlap)

                if l=='Length':
                    value=i.get(l)
                    avg_length+=value
        avg_length=avg_length/tot_ann

        # overlap similarity = length of overlap list / length of average annotated line
        overlap_similarity= len_overlap / avg_length
```

Di seguito i risultati ottenuti:

Results:
****
****

Similarity for the word "COURAGE": 0.43971631205673756
Overlap list for "COURAGE": ['face', 'fear']

Similarity for the word "PAPER": 0.4881889763779528
Overlap list for "PAPER": ['material', 'cellulose']

Similarity for the word "APPREHENSION": 0.2440944881889764
Overlap list for "APPREHENSION": [['']]

Similarity for the word "SHARPENER": 0.7948717948717948
Overlap list for "SHARPENER": ['pencils', 'blade', 'sharpen']

2.

Content-form-relations

Nell'esercizio veniva chiesto di fare una caratterizzazione delle definizioni in wordnet tramite uno studio di forma, contenuto e relazioni.

1) ***Studio di forma***:

Per quanto riguarda lo studio di forma ho calcolato la lunghezza media delle definizioni in wordnet, insieme alla lunghezza totale delle definizioni. Per fare ciò ho iterato su tutte le definizioni di wordnet tramite il metodo get_average_len_def() per poi dividere la somma per il numero di definizioni totali.

2) ***Studio di contenuto***:

Nello studio di contenuto vogliamo trovare la profondità media delle definizioni in wordnet.
Per prima cosa ho iterato su tutte le definizioni dei sensi di wordnet. Dopodiché dopo aver effettuato un passo di preprocessing, dove vengono eliminati segni di punteggiatura e stopwords, per ogni elemento della nuova lista, si va a costruire il contesto(create_context()).
Nel contesto si prendono tutti i sensi per quella parola e si crea il context, trovando i relativi gloss e example. Viene creato il contesto anche per iponimi e iperonimi. Una volta trovato i contesti, si può cercare il miglior senso per quel dato contesto.
Nel metodo bag_of_words, dopo aver effettuato il preprocessing delle definizioni, si trova l'intersezione tra l'insieme delle definizioni preprocessate con l'insieme che è stato trovato nel context. Dopodiché si prende la lunghezza della lista e si cerca quella max. Una volta ottenuta, di restituisce il best sense, ovvero il senso che massimizza tale lunghezza.
Una volta trovato il best sense attraverso il metodo bag_of_words, si calcola la profondità di tale senso nella tassonomia di wordnet.
Infine, viene restituita la lunghezza media delle definizioni, calcolando il quoziente tra la somma delle avg_definitions e il numero di best sense trovati.

3)***Studio di relazioni:***

Per prima cosa nello studio di relazioni si è creato un dizionario contenente come chiave tutte le relazioni presenti in wordnet, ovvero: hypernym, hyponyms, meronyms, holonyms, synonyms e antonyms. A ciascun chiave inizialmente è associato un oggetto vuoto.
Dopodiché si estraggono nuovamente le definizioni per ciascun synset, e per ogni synset e relazione si ricorre a un metodo chiamato update_relation_set(), che effettua una distinzione tra relazioni transitive e intransitive. Nel caso una relazione sia intransitiva, si va a calcolare tutte le occorrenze che tale relazione ha in wordnet in modo ricorsivo. Anche qui ci si ferma se raggiunto il numero "tetto" di iponimi fissato. Alla fine, dopo aver finito di contare le occorrenze, si salva il risultato in una lista. Per quanto riguarda le relazioni transitive si va a vedere se il senso preso in considerazione ha antonyms e synonyms presenti nelle definitions, se sì lo salvo in una lista.
Una volta finito sto processo si va a popolare il dizionario con i risultati ottenuti.

Risultati:

```
Average length of definitions in Wordnet: 10.531468055992317
Length of all definitions in Worndet: 117659
Average height of definitions from Wordnet Root: 4.3789123627836375
len of relations of type  HYPERNYMS  :  69201  of  117659  --->  0.588148802896506
len of relations of type  HYPONYMS  :  1088  of  117659  --->  0.009247061423265538
len of relations of type  MERONYMS  :  731  of  117659  --->  0.006212869393756534
len of relations of type  HOLONYMS  :  298  of  117659  --->  0.00215028174640274
len of relations of type  ANTONYMS  :  694  of  117659  --->  0.005898401312266805
len of relations of type  SYNONYMS  :  8312  of  117659  --->  0.07064482954980O7
```

3.

Hanks theory

1. Sono stati utilizzati i verbi:
   - build
   - love
   - eat

2. Per recuperare n frasi in cui viene usato il verbo selezionato è stato utilizzato il corpus brown. Dato che la forma della parola relativa al verbo cambia in base alla coniugazione di quest'ultimo è stato necessario l'utilizzo del lemmatizzatore di wordnet. Iterando su tutte le parole di una singola frase, per ogni parola viene calcolato il lemma e se è uguale alla forma base del verbo scelto la frase viene salvata.

3. Per ogni frase, con l'ausilio di spacy, viene effettuato il pos + parsing, ottenendo i vari token collegati tramite dipendenze sintattiche. Iterando su tutti i token è possibile estrarre il **soggetto** e il **complemento oggetto** relativi al verbo della frase, in quanto sono token figli del token del verbo a cui sono legati.

4. **Soggetto e Complemento oggetto** vengono mappati con i relativi super sensi di Wordnet nel seguente modo:

- ***Nome proprio***: person
- ***Pronome*** : [I, you, she, he, we, they] : person oppure [It] : entity
- ***Parola generica***: viene utilizzato il WSD che data la frase e il termine generico tenta di restituire il senso del termine

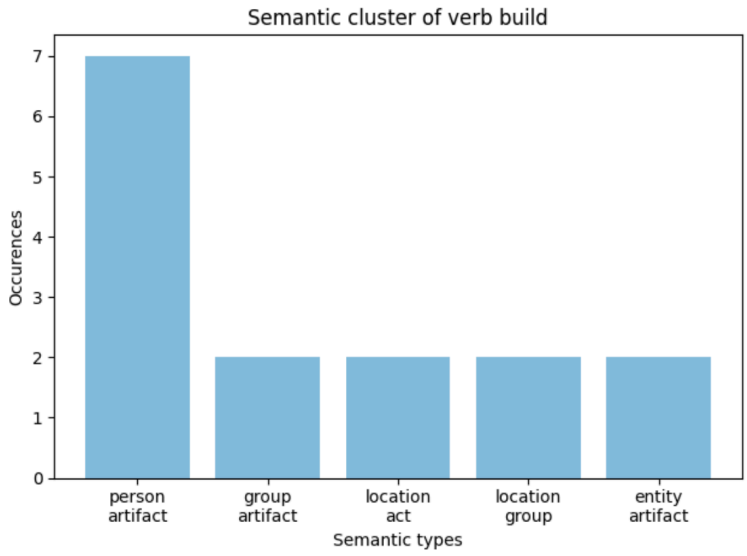
5. I supersensi di soggetto e complemento oggetto rappresentato rispettivamente i filler1 e filler2 dei verbi (in quanto tutti e 3 i verbi hanno valenza 2), le combinazioni dei tipi semantici sono salvati nei semantic cluster.

- **Semantic cluster** contiene tutte le coppie **super_sense_subj:super_sense_dobj** estratte dalle frasi.

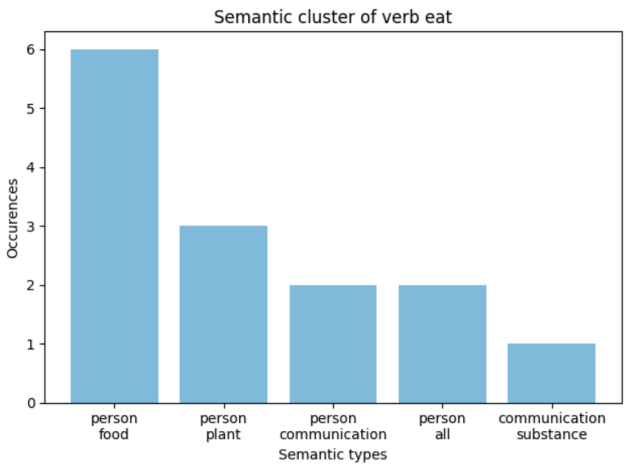
5. Vengono calcolate la frequenze dei coppie supersensi, ottenendo così per Filler1 e Filler2 i supersensi più frequenti.

I **cluster semantici** ottenuti sono visibili nei grafici sottostanti con le relative frequenze

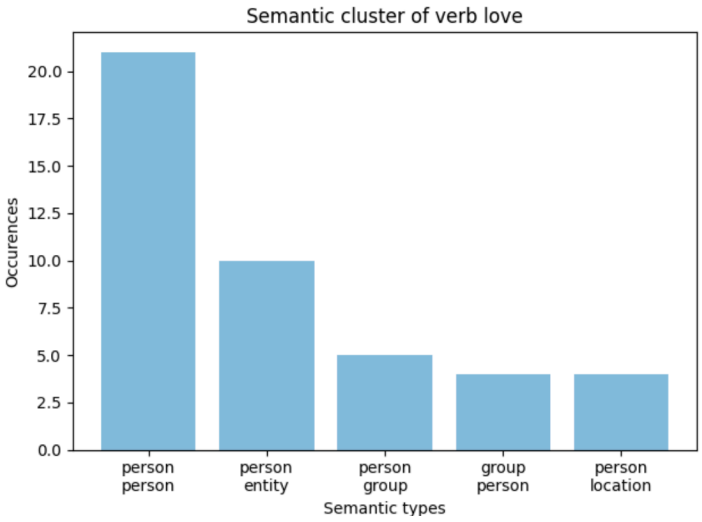
Semantic type for **Build**:



Semantic type for **Eat**:



Semantic type for **Love**:



Come si può vedere la coppia più frequente è quella più significativa per il significato del verbo:

- (person:person : 20) per il verbo amare
- (person:food : 7) per il verbo mangiare
- (person:artifact : 7) per il verbo costruire

Alcune coppie di semantic type, quella con frequenza minore, possono essere poco significative per il verbo in esame, ma questo può essere dovuto all'accuratezza del WSD che è pari al **68%**.

4. Content-to-form

Implementazione:

Input : Lista definizioni per il termine **w**

Output: **w** o un concetto semanticamente legato a w

1. Estrazione termini rilevanti e soggetti/aggettivi dalle definizioni
2. Estrazione dominio e contesto

- **dominio**: possibili iperonimi usati per descrivere w con relativi aggettivi
- **contesto**: termini rilevanti presenti nelle definizioni

3. Calcolo delle frequenze dei domini e contesti in modo da operare unicamente su quelli più frequenti
4. Ricerca in Wordnet su ogni termine presente nel **dominio**, dato un termine nel dominio si guardano tutti i sensi associati al synsets di wordnet e per ogni senso si valutano tutti gli iponimi
5. Viene calcolata la similarità tra la definizione del synset e la stringa ottenuta dalla concatenazione di tutti i termini nel **contesto**. Il synset con il valore massimo di similarità viene ritornato.

1. Estrazione termini rilevanti

Da ogni definizione vengono estratti:

- **Soggetti**: effettuando Pos Tagging + Parsing su ogni definizione
- **Termini rilevanti**: tutti i termini presenti nelle definizioni eliminando eventuali stopword e simboli di punteggiatura.

2. Estrazione dominio e contesto

Vengono costruite 2 liste:

- contex = i termini rilevanti della definizione
- domain = soggetto e/o aggettivi relativi alla definizione corrente unito a tutti i domini (estratti da wordnet) legati ai termini presenti nell'insieme delle parole rilevanti.

I punti 1 e 2 vengono eseguiti su ogni definizione per il termine w. Gli insiemi contex e domain sono cumulativi, ad ogni iterazione su una definizione diversa vengono estesi con i nuovi termini rilevanti e i nuovi domini ottenuti dalla definizione corrente.

3. Calcolo delle frequenze

Date le liste contex e domain vengono calcolati 2 insiemi:

- context = i contesti più frequenti sulla frequenza dei termini in contex
- candidate_genus = i domini più frequenti sulla frequenza dei termini in domain

Vengono mantenuti solo i domini e contesti più frequenti, in base ai seguenti parametri:

- max_term_in_context
- max_genus

4 Ricerca in Wordnet

Viene costruita la stringa:

- s_context = come la concatenazione dei lemma di ogni parola nell'insieme context.

La ricerca è di tipo top-down, l'algoritmo itera su ogni termine **w** presente nell'insieme candidate_genus, si valutano tutti i sensi associati ad w e tutti gli iponimi di ogni senso.

5 Similarità

La valutazione di ogni senso avviene calcolando la similarità tra la definizione del senso e la stringa s_context, la misura di similarità adottata utilizza la cosin similarity tra vettori embedding relativi ai termini.

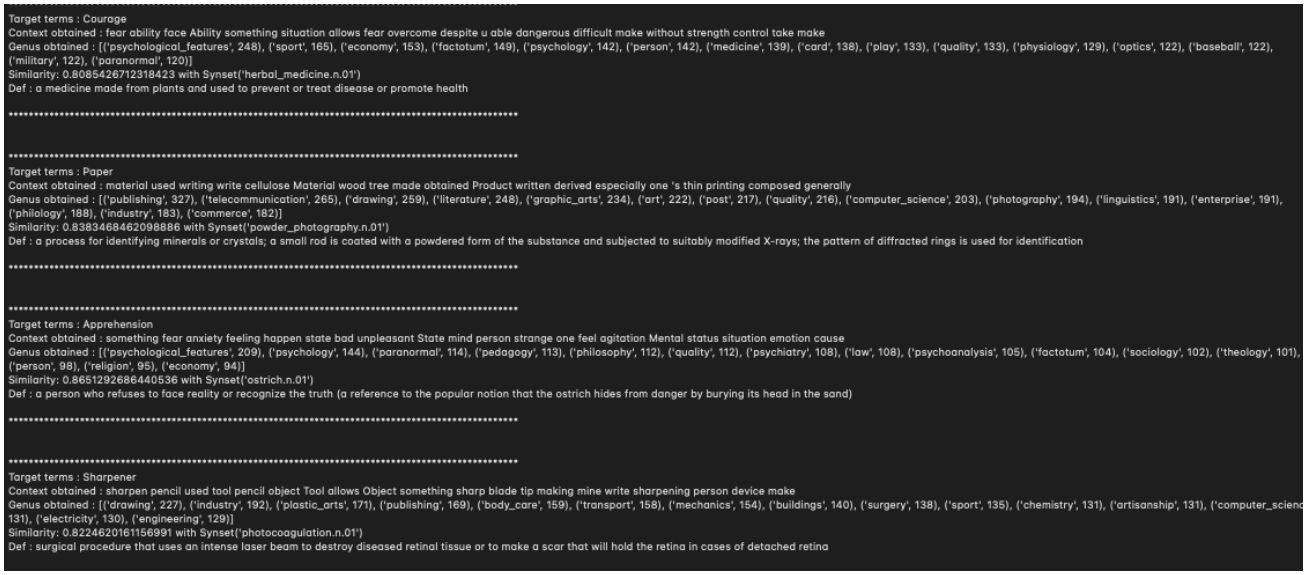
- Lo score associato a ciascun synset è il valore della similarità.

Durante la ricerca vengono aggiornati 2 valori:

- best_score = punteggio da 0 a 1
- best_synset = synset relativo allo score

Dopo avere iterato su ogni termine nell'insieme candidate_genus viene ritornato il synset con lo score maggiore.

Risultati:



I risultati hanno avere senso se si considera il fatto che ci sono state delle annotazioni con errori sintattici, per quanto riguarda l'inglese.

Seconda Parte

1. Text Segmentation

Il task è stato realizzato eseguendo i seguenti step:

1. ***Windowing + Tokenizing***
   - Trasformazione del testo in un insieme di **N** frasi.
   - Le frasi vengono raggruppate in finestre di lunghezza fisse, specificata dal parametro size_windows, ottenendo così **N/size_windows** finestre della stessa lunghezza.
   - Ogni finestra viene 'tokenizzata' trasformando l'insieme di frasi al suo interno in una lista di parola rilevanti.
2. ***Cohesion ranking***
   - Per ogni finestra i-esima viene valutata la similarità intra gruppo.
   - La media fra la similarità tra la finestra i-1 e i e quella fra la finestra i e i +1
3. ***Searching of Break Points***
   - Ricerca dei punti a bassa coesione circondanti da quelli ad alta coesione.

Saranno ora descritti i 3 step con maggiore dettaglio.

1. Windowing + Tokenizing

Il testo viene letto da file e trasformato in una lista di frasi, successivamente le frasi vengono raggruppate in finestre di lunghezza fissa e la cardinalità di ogni finestra è data dal parametro size_windows. Lo step successivo è quello di estrarre i termini rilevanti in ogni finestra filtrando stop_word e rimuovendo i vari simboli di punteggiare.
La struttura dati risultati è una lista di finestra in cui in ogni elemento i-esimo sono presenti i termini rilevanti per la finestra i-esima.

2. Cohesion ranking

Lo scopo di questo step è quello di calcolare la coesione di una finestra rispetto a quelle adiacenti (in quella precedente e in quella successiva), sulla base dei termini rilevanti presenti in ogni finestra.
Il valore di coesione di una finestra è dato da quanto le parole che occorrono nella finestra sono semanticamente simili alle parole che occorrono in quelle adiacenti, di conseguenza per ogni coppia di finestre adiacenti è stata calcolata la similarità semantica per coppia di parole presenti nelle finestre. Date le finestre **A** e **B** il valore di coesione è dato dalla seguente formula:

$$Coesion(A,B) = \frac{\sum_i^{|A|} \sum_j^{|B|} sim(w_i, w_j)}{|A||B]}$$

Dove le due sommatorie iterano su ogni combinazione di parole presenti nelle finestre e **sim(a,b)** è il valore di similarità fra parola a e b. E' stata utilizzata come misura di similarità fra parole la **Weighted Overlap**: basato sulla versione **Lexical** di NASARI.
Il valore di similarità per la finestra i-esima viene calcolata come **media** dei seguenti valori:

- similarità tra la finestra i e i + 1
- similarità tra la finestra i e i - 1

Weighted Overlap
Il programma effettua parsing della versione Lexical di NASARI mantenendola in memoria per accedere alla rappresentazione vettoriale di una parola quando viene coinvolta nel calcolo della misura di similarità.
Per ogni finestra (corrente, precedente e successiva) vengono estratti i relativi vettori Nasari, ovvero una lista di vettori relativi a tutte le parole rilevanti presenti nella finestra. Dato i vettori v1 e v2 viene calcolata l'intersezione tra i 2 e valutato il rank degli elementi comune, la formula è la seguente:

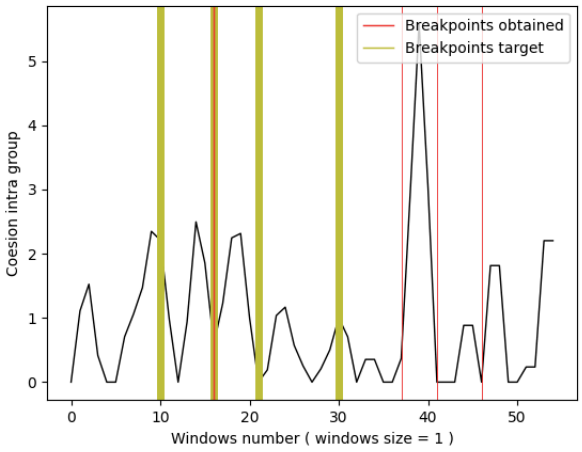
$$WO(v_1, v_2) = \frac{\sum_{q \in O} \left(rank(q, v_1) + rank(q, v_2)\right)^{-1}}{\sum_{i=1}^{|O|} (2i)^{-1}}$$

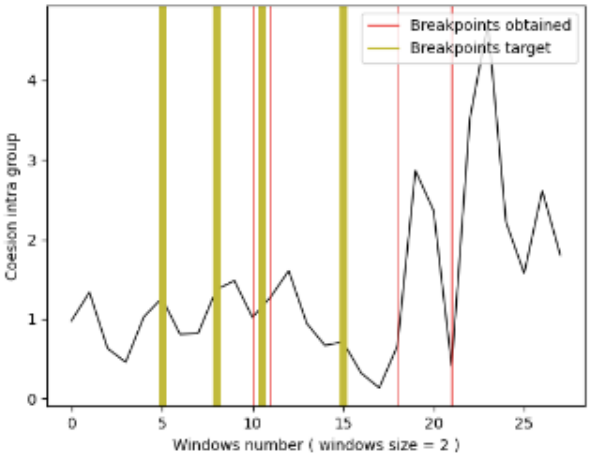
3. Search Break Point

Per determinare i punti in cui dividere il testo, in modo da determinare i paragrafi, si scorre la lista contenente i valori di coesione similarities. Una finestra è un buon candidato per un punto di split point se il suo valore di coesione è minore rispetto la media dei valori totali e se la somma delle similarità della finestra successiva e precedente è elevata.
Si cerca il miglior candidato, ovvero fra tutte le finestre con un valore di similarità basso (sotto la media) si prende quella, la cui somma di similarità delle finestre adiacente è massima. L'algoritmo procede iterativamente andando ad estrarre ad ogni iterazione il miglior minimo, la condizione di stop è che non ci sono più 'migliori minimi' da estrarre oppure è riuscito a trovare il numero target di split points.

Risultati

Il testo utilizzato è un breve biografia dell'atleta Muhammad Ali, presente in utils/muhammadali.txt.
Sono stati effettuati diversi esperimenti variando la dimensione delle finestre. Nella variabile break_point_sentence_target ci sono gli indici delle frasi che indicano il cambio di discorso all'interno del testo.



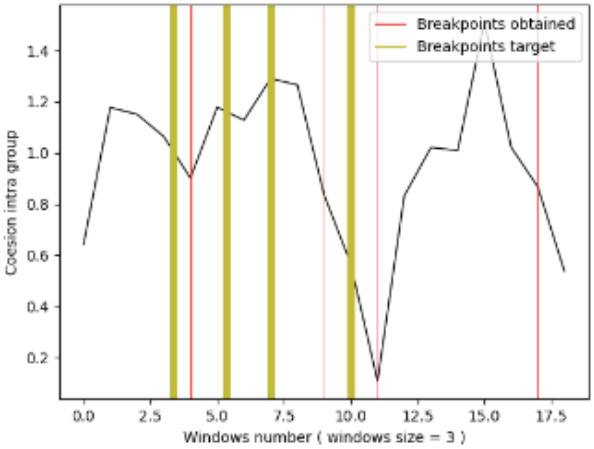


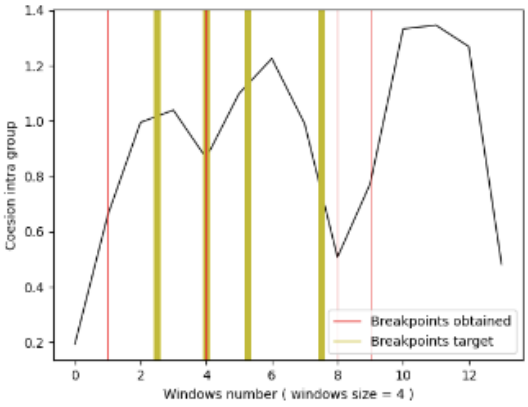
Di seguito sono indicati i risultati dei vari esperimenti:

- **riga rossa : split points ottenuto**
- **riga gialla: split point target**

Durante l'esecuzione del programma gli indici nella variabile break_point_sentence_target vengono divisi per il numero di frasi per finestra in modo da identificare a grandi linee in quale finestra è presente la frase che indica il cambio del discorso.

Di conseguenza i valori di break_point_sentence_target possono anche essere decimali, e se nel grafico la linee gialla è molto vicina a quella rossa il programma, in questa particolare configurazione delle finestre, non poteva arrivare ad una soluzione più corretta. (In quanto la frase che genera il salto del discorso è interna ad una finestra).





Come si può vedere dai grafici, ci si avvicina maggiormente al target nel caso in cui la windows_size è 4.

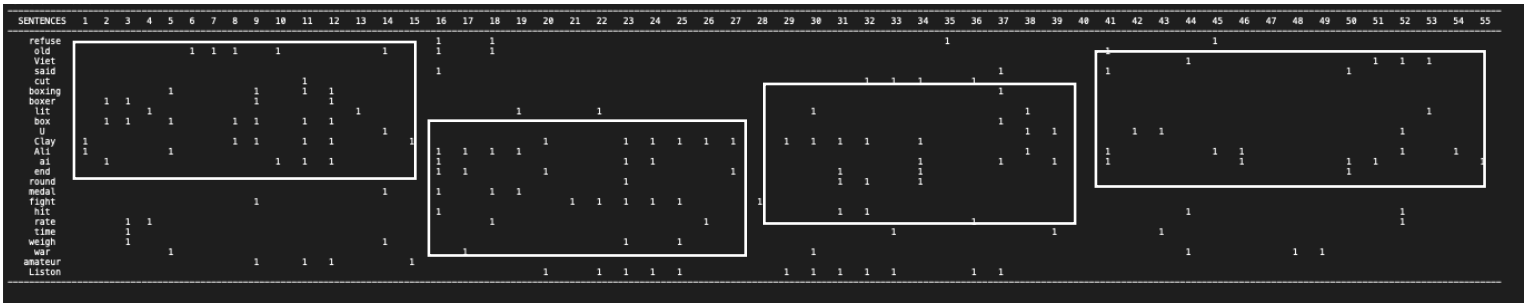
E' anche stato realizzato un metodo che dal testo costruisce una matrice dove sulle:

- righe : sono indicati i termini rilevanti (non stop word e non numeri) con occorrenze totale nel testo maggiore di 3
- colonne : indici delle frasi all'interno del testo
- celle(i, j): occorrenza parola della riga i nella frase con indice j

Come si può vedere in figura, in questo testo ci sono alcuni punti in cui i cambi di discorso sono evidenti in quanto le parole usate variano molto da una finestra ad un'altra. Ad esempio, dalla frase 16 alla 27 (seconda finestra) le prime 6 parole non vengono più

utilizzate. NOTA: per eseguire il programma è necessario che in utils ci sia la versione lexical di **NASARI** (in questa versione sono presenti 10 feature per ogni item) disponibile al seguente url:

- https://goo.gl/85BubW



2. **Topic Modeling**

L'esercizio ha come consegna quella di sviluppare un sistema che, sfruttando la tecnica del topic modeling, parte da un testo e ci restituisce una lista di topics.

Il corpus è stato preso da *The Chapel - Short Story* di Josef Essberger.

L'algoritmo funziona come segue:

1) Si effettua preprocessing di tutti i paragrafi del corpus.

2) Si attribuisce ad ogni termine del testo un id univoco sfruttando dictionary della libreria gensim, e si crea una rappresentazione bag-of-words attraverso il doc2bow presente in gensim. Questo ci permette di avere una rappresentazione sparsa delle occorrenze di ciascuna parola nel paragrafo.

3) Si crea il modello LDA grazie alla funzione già presente di Gensim, e per ogni paragrafo si estraggono 10 topics.

4) Si estrae una rappresentazione dei topics, salvata in una lista di dizionari che contiene le parole e le probabilità di ciascun termine nel modello LDA.

Esempio output:

[{'words': ['ginnie', 'ravi', 'kirjani', 'said', 'like', 'man', 'little', 'marry', 'even', "n't"], 'probs': [0.012756876, 0.009960511, 0.005482784, 0.005272354, 0.004972939, 0.0049362057, 0.0049296813, 0.0045768335, 0.004571298, 0.0043619988]},…]